UNIVERSITY OF TWENTE.

Module 8: Programming Paradigms
Logic Programming
part 1: facts, rules & queries

Sebastiaan J.C. Joosten (slides in part: Jaco van de Pol)

14 May 2019

Schedule LP

- Three lectures: May 14th, 16th, 22nd (block 4,5)
- Visit them: not all I say is on the slides.
- Idea:
  - 1 hour colstruction (lecture + instruction)
  - 3 hours practical exercises (signed off, in pairs)
  - You should already start with the project in block 5.

- LP project: May 21 - May 28 (block 5)
  - Work on the project (handed in, in pairs)
  - Last session on May 28, deadline on May 29th.
- All material:
  - Canvas, Block 4,5 (reader, labfiles)
  - Tooling (SWI-Prolog)

Overview

1. Facts, rules & queries by Example:
   - Example: The Royal Family (exercise 1)

2. Structured objects with function symbols (terms)

3. Problem solving by Example:
   - Example: Monkey and Banana (exercise 2)

4. Lists: recursion and backtracking
   - Lists and Tree-sort (exercise 3)

Prolog – logic programming

A Prolog program consists of set of ***clauses*** (facts and rules).

You "run" Prolog by asking a ***query***.

Prolog tries to find a ***proof*** of the query from the facts and rules.

It does so by systematic search, depth-first, using ***backtracking***.

Here is your first Prolog program:

```
a:-b.       % a follows from b
a:-c, d.    % a follows from c and d
c.          % c is a true fact
d:-c.       % d follows from c
```

When asked to prove the query a:

- First, Prolog tries to prove b, which fails.
- Then it is tried to prove c and d.
  - c is a fact, so this holds
  - for d, we must prove c (again), this will succeed.

Predicates, terms, variables

Predicates (like a,b,c on the previous slide) can have arguments:

Every argument is a term. For now, we a term can be:

1. An atom: lower case identifier, denotes a fixed object or number
2. A variable: upper case identifier, can be bound to any object

Given Clauses (the program consists of these two facts):

age(john,33).

age(peter,21).

Queries:	**Results:**
?- age(john,33).	**yes / true**
?- age(john,21).	**no / false**
?- age(X,21).	**X=peter**
?- age(John,21).	**John=peter**
?- age(john,X).	**X=33**
?- age(X,Y).	**X=john, Y=33;**
	X=peter, Y=21;

Prolog returns either **no**, or **yes** with a list of **bindings** for the logical variables

Example: Royal family (fact database)

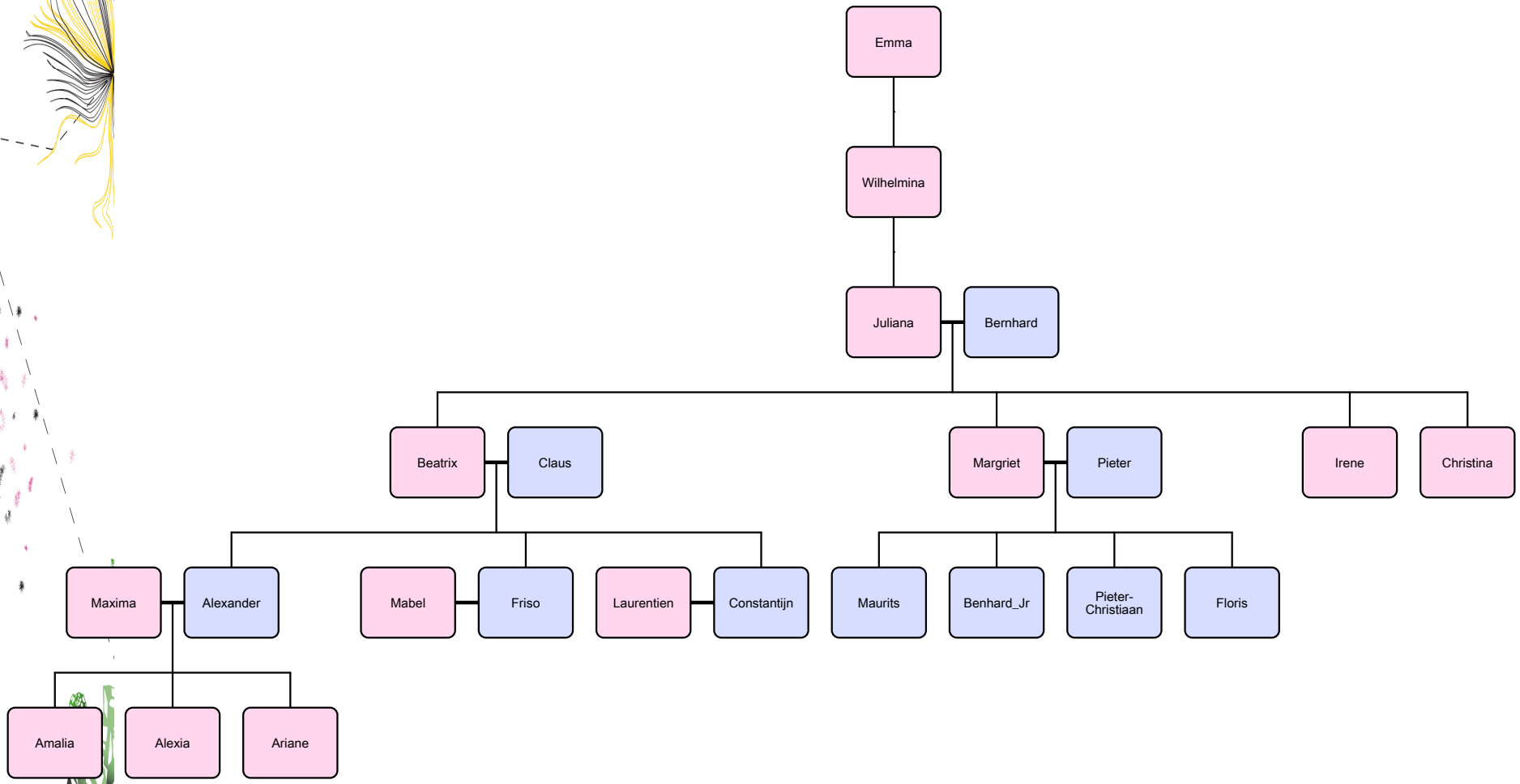
```
mother(emma,wilhelmina).
mother(wilhelmina,juliana).
mother(juliana,beatrix).
mother(juliana,margriet).
mother(juliana,irene).
mother(juliana,christina).
mother(maxima,amalia).
mother(maxima,alexia).
mother(maxima,ariane).
…
husband(bernhard,juliana).
husband(claus,beatrix).
husband(pieter,margriet).
husband(alexander,maxima).
```

```
female(irene).
female(amalia).
…
female(X) :- mother(X,_).
female(X) :- husband(_,X).

male(maurits).
male(floris).
…
male(X) :- husband(X,_).
```

Note, we have used don't care variables: _

Royal Family



Interpretation of clauses

mother(maxima,amalia) is interpreted (by us, not by Prolog) as: *maxima is the mother of amalia*.

father(X,Y) :- husband(X,Z), mother(Z,Y).

means: *for each X, Y and Z: X is the father of Y, if X is the husband of Z and Z is the mother of Y*.

child(X,Y) :- father(Y,X).

child(X,Y) :- mother(Y,X).

means: *for each X and Y: X is a child of Y, if Y is the father of X, or Y is the mother of X.*

Alternative (; means "or")

child(X,Y) :- father(Y,X) ; mother(Y,X).

Clauses: facts, rules, queries

- A ***clause*** is of the form **h :- b1, …, bn**.
  - Here h, b1, …, bn are called ***literals***.
  - Facts, rules and queries are all clauses:

- A ***rule*** is a clause with a non-empty head and body:
  - **h** is the ***head*** of the rule
  - **b1, …, bn** is the ***body*** of the rule

- A ***fact*** is a clause with an empty body:
  - **h.**

- A ***query*** is a clause with an empty head:
  - **:- b1, ... , bn.** Sometimes also written as: **?- b1, ..., bn**.

Exercise (1)

1. Download SWI-prolog from http://www.swi-prolog.org.
2. Download the Prolog program "royal_family_facts" from Canvas. (in the zip: PP->Materials->block5)
3. Start Prolog with this program.
4. Evaluate each of the following queries, first by inspecting the program, then by using Prolog.
   - **a:** **?- mother(juliana,Beatrix).**
   - **b:** **?- female(X).**
   - **c:** **?- husband(X,Y).**
5. Write a predicate **grandfather/2**.
6. Write a predicate **sister/2**.

Exercise (2)

Solutions:

grandfather(X,Y):- child(Y,Z),child(Z,X),male(X).

sister(X,Y) :- child(X,P), female(X), child(Y,P), not(X=Y).

This exercise is continued later during exercise class,

By writing and testing predicates like grandson, aunt, cousin...

Let's write a recursive program:

ancestor(X,Y) :- child(Y,X).

ancestor(X,Y) :- child(Y,Z) , ancestor(X,Z).

Try:

?- ancestor(X,amalia).

?- ancestor(X,maurits) , ancestor(X,alexia).

Will this predicate always "terminate"? Why? When?

The paradigm of Logic Programming, so far

- Allows a very free format:
  - We didn't have to provide any declaration, just facts and rules
  - The only "convention" is the distinction upper/lower case
- No pre-determined computation order, you can freely mix "forward" and "backward" computations
- That is: Prolog works with **relations**, not with **functions**
- Beware: Prolog uses a ***closed world assumption***
  - **?- husband(X,emma).** ***No.***
  - *In reality she did have a husband, but he wasn't in the database*

Overview

1. Facts, rules & queries by Example:
   - Example: The Royal Family (exercise 1)

2. Structured objects with function symbols (terms)

3. Problem solving by Example:
   - Example: Monkey and Banana (exercise 2)

4. Lists: recursion and backtracking
   - Lists and Tree-sort (exercise 3)

Structured data

- Data can be structured, like in

  date(23, may, 2016)

  triangle(point(4,5),point(5,9),point(13,3))

- In general, terms consist of:
  - **atoms (lower-case identifiers, numbers)**
  - **variables (upper-case identifiers)**
  - **function application f(t1,…,tn), where ti are terms**

- Prolog allows infix notation, like 1+3, 1=<5
  - Here + and =< are the function symbols

- Special list notation: [], [Head|Tail]
  - [] is a constant, and [_ | _] is a binary function symbol

Terms and unification

- Variables can be used for “pattern matching”
- For instance:
  - **?- date(23,may,2016) = date(Day,Month,Year).**
    Yes. [Day=23, Month=may, Year=2016]
  - **?- point(3,4)=point(X,_).**
    Yes. [X=3]
  - **?- f(X,f(Y)) = f(f(Z),Z).**
    Yes. [X=f(f(Y)), Z=f(Y)]
- In general, **term1 = term2** checks if these two terms can be made equal by substituting terms for the variables.
- Prolog always computes *the most general unifier*

Lists and the typical list recursion

Lists are just terms, using a special function-notation.

Lists in Prolog are inhomogeneous: elements can be any terms.

Notation: **[] is the empty list**

[X] is the list with single element X

[X|Y] is the list with head X and tail Y

[X,Y|Z] is the list with elements X and Y and tail Z

The predicate member/2 tests if an element is in the list:

```
member(X,[X|_]).
member(X,[_|L]) :- member(X,L).
```

The predicate append/3 concatenates two lists

```
append([],Xs,Xs).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```

Note: this function can be used in many ways!!

Overview

1. Facts, rules & queries by Example:
   - Example: The Royal Family (exercise 1)

2. Structured objects with function symbols (terms)

3. Problem solving by Example:
   - Example: Monkey and Banana (exercise 2)

4. Lists: recursion and backtracking
   - Lists and sorting (exercise 3)

Monkey gets Banana

- Prolog is good at problem solving
- We will go through a very simple planning problem:

- A monkey wants a banana hanging from the ceiling
- For this, he must walk to a box, push it, climb it

- We model *"the state of the world"* a structure:
  state(Monkey, Position, Box, Has)
  - ***Monkey*** is the location of the monkey (atdoor,atwindow,middle)
  - ***Position*** is the position of the monkey (onfloor, onbox)
  - ***Box*** is the location of the box (atdoor, atwindow, middle)
  - ***Has*** indicates if the Monkey has the banana (has, hasnot)

The actions of the Monkey

The monkey can perform several actions, between two states of the form **state(Monkey,Position,Box,Has):**

- **grab** - **grab the banana**
- **climb** - **climb on top of the box**
- **walk(Pos1,Pos2)** - **walk to from position 1 to 2**
- **push(Pos1,Pos2)** - **push the box from Pos 1 to 2**

- The actions are only possible under certain conditions, *which we must define by rules in the program*

- Will the monkey be able to grab the banana? *which we must formulate as a query*

Goal and Moves

- The initial state can be described by the term:
  state(atdoor,onfloor,atwindow,hasnot)

- The goal states can be described by the term:
  state(_,_,_,has)

- We describe the possible actions by a predicate move/3:
  move(State1,Action,State2).
- If the monkey is on the floor at the box, it can climb it:

```
move( state(Pos, onfloor, Pos, Has),
      climb,
      state(Pos, onbox, Pos, Has) ).
```

- **Exercise:** describe the other actions: **grab, walk, push**

A generic problem solver in Prolog

- We can now write a very simple and general solver:

```
solve(state(_,_,_,has)).
solve(State1):- move(State1,Move,State2), solve(State2).
```

- Now Prolog can automatically solve the problem:

```
?- solve(state(atdoor,onfloor,atwindow,hasnot)).
Yes.
```

- **Exercise 2 today:**
  - Finish the whole puzzle.
  - Let Prolog compute a concrete sequence of actions.

Overview

1. Facts, rules & queries by Example:
   - Example: The Royal Family (exercise 1)

2. Structured objects with function symbols (terms)

3. Problem solving by Example:
   - Example: Monkey and Banana (exercise 2)

4. Lists: recursion and backtracking
   - Lists and Tree-sort (exercise 3)

Exercise

1. Write a predicate **quicksort**, which sorts a list using the well-known quicksort-algorithm.

2. Write a predicate **slowsort**, which sorts a list by searching through all permutations of the list for a sorted one.

We will use the built-in comparison operator:

M =< N

Solution for slowsort

```
slowsort(X,Y) :- perm(X,Y), sorted(Y).

sorted([]).
sorted([X]).
sorted([X,Y|Z]) :- X=<Y, sorted([Y|Z]).

perm([],[]).
perm([X|Y],Z) :- perm(Y,A), insert(X,A,Z).

insert(X,A,[X|A]).
insert(X,[K|S],[K|R]) :- insert(X,S,R).
```

Solution for quicksort

```
quicksort([],[]).
quicksort([A|B],C) :-
   split(A,B,D,E),
   quicksort(D,F),
   quicksort(E,G),
   append(F,[A|G],C).

% split(A,B,D,E): Split list B in D and E by comparing with A

split(X,[],[],[]).
split(X,[A|B],[A|C],D) :- A=<X, split(X,B,C,D).
split(X,[A|B],C,[A|D]) :- A>X, split(X,B,C,D).
```

General approach to Prolog programming

1. Declarative step: Model the problem
   - Decide which structures to use (function symbols)
   - Define the important relations (predicates)

2. Algorithmic step: Write a general solver
   - Independent from the problem

3. Procedural thinking: Make it work
   - Take into account termination (order of rules)
   - Take into account efficiency, search strategy, pruning

Largest pitfall: Start too early thinking procedurally